Zend_Context - Peter Kovacs

<ac:macro ac:name="unmigrated-inline-wiki-markup"><ac:plain-text-body><![CDATA[

<ac:macro ac:name="unmigrated-inline-wiki-markup"><ac:plain-text-body><![CDATA[

Zend Framework: Zend_Context Component Proposal

Proposed Component Name	Zend_Context
Developer Notes	http://framework.zend.com/wiki/display/ZFDEV/Zend_Context
Proposers	Peter Kovacs
Revision	0.1 - 29 January 2008: Started proposal process 0.2 - 24 Febuary 2008: Proposal ready for review 0.3 - 11 June 2008: Added type casting (wiki revision: 26)

Table of Contents

1. Overview
2. References
3. Component Requirements, Constraints, and Acceptance Criteria
4. Dependencies on Other Framework Components
5. Theory of Operation
6. Milestones / Tasks
7. Class Index
8. Use Cases
9. Class Skeletons

1. Overview

Zend_Context implements Dependency Injection.

2. References

- Martin Fowler: Inversion of Control Containers and the Dependency Injection pattern
- MSDN Magazine: Desing Patterns: Dependency Injection
- Spring Framework
- Spring.NET

3. Component Requirements, Constraints, and Acceptance Criteria

- It **will** read XML or php context files.
- It **will** not read ini files.
- It **will** provide a static class for getting objects.
- It **will** provide methods for creating components from code.
- It **will** provide interface for reading variables from config files

4. Dependencies on Other Framework Components

- Zend_Exception
- Zend_Loader

5. Theory of Operation

Configure the objects and their relations/dependencies of an application's context in a simple way (php or xml files) and load them easily. For dynamic component creation and loading there will be a small interface.

There is a more complex approach in: http://framework.zend.com/wiki/display/ZFPROP/Zend_Di+-+Dependency+Injection+Container

6. Milestones / Tasks

- Milestone 1: [DONE]Design
- Milestone 2: [DONE]Use cases
- Milestone 3: [DONE]Unit tests
- Milestone 4: [DONE]Api doc
- Milestone 5: [DONE]Proposal
- Milestone 6: Work on design, use cases and unit tests from comments
- Milestone 7: Documentation
- Future: New ideas (import, inheritance, scope, validation)

7. Class Index

- Zend_Context (static class)
- Zend_Context_Exception
- Zend_Context_Component
- Zend_Context_Component_Interface
- Zend_Context_Loader (factory class)
- Zend_Context_Loader_Array
- Zend_Context_Loader_Exception
- Zend_Context_Loader_Xml
- Zend_Context_Property

8. Use Cases

UC-01: XML context file for use cases (test.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<context  xmlns="http://www.zend.com/schema/context"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="
http://www.zend.com/schema/context http://www.zend.com/schema/context/context.xsd">
    <property value="contextProperty"/>
    <registry>Zend_Registry</registry>
    <components>
        <component>
            <id>baseSql</id>
            <class>Zend_Db</class>
            <default>yes</default>
            <factory>factory</factory>
            <constructor>
                <arg>PDO_Pgsql</arg>
                <arg>
                    <array>
                        <elem key="host">db.example.com</elem>
                        <elem key="port" type="int">1234</elem>
                        <elem key="username">dbuser</elem>
                        <elem key="password" value="secret"/>
                        <elem key="dbname" value="db"/>
                    </array>
                </arg>
            </constructor>
        </component>
        <component>
            <id>logWriter</id>
            <class>Zend_Log_Writer_Stream</class>
            <constructor>
                <arg value="php://output"/>
            </constructor>
        </component>
        <component id="log" class="Zend_Log" default="yes">
            <methods>
                <method name="addWriter">
                    <arg ref="logWriter"/>
                </method>
            </methods>
        </component>
        <component id="contextProperty" class="Zend_Context_Property">
            <constructor>
                <arg value="include_path:../tests/Zend/Context/_files/test1.ini"/>
            </constructor>
        </component>
    </components>
    <aliases>
        <alias key="pgSql">baseSql</alias>
    </aliases>
</context>
```

UC-02: php context file for use cases (test.php)

```
<?php
return array(
            "property"=>"contextProperty"
            ,"registry"=>"Zend_Registry"
            ,"components"=>array(
                "baseSql"=>array(
                    "class"=>"Zend_Db"
                    ,"default"=>"yes"
                    ,"factory"=>"factory"
                    ,"constructor"=>array(
                        array("value"=>"PDO_Pgsql")
                        ,array(
                            "value"=>array(
                                array("key"=>"host","value"=>"db.example.com")
                                ,array("key"=>"host","type"=>"int","value"=>"1234")
                                ,array("key"=>"username","value"=>"dbuser")
                                ,array("key"=>"password","value"=>"secret")
                                ,array("key"=>"dbname","value"=>"db")
                            )
                        )
                    )
                )
                ,"logWriter"=>array(
                    "class"=>"Zend_Log_Writer_Stream"
                    ,"constructor"=>array(
                        array("value"=>"php://output")
                    )
                )
                ,"log"=>array(
                    "class"=>"Zend_Log",
                    "default"=>"yes"
                    ,"methods"=>array(
                        "addWriter"=>array(
                            array("ref"=>"logWriter")
                        )
                    )
                )
                ,"contextProperty"=>array(
                    "class"=>"Zend_Context_Property"
                    ,"constructor"=>array(
                        array("value"=>"include_path:../tests/Zend/Context/_files/test1.ini")
                    )
                )
            )
            ,"aliases"=>array("pgSql"=>"baseSql")
        );
```

UC-03: Loading an xml file and getting an object

```
$loader = Zend_Context_Loader::factory("xml","test.xml");
Zend_Context::load($loader);
$db = Zend_Context::getObject("baseSql");
```

UC-04: Loading a php file and getting an object by class

```
$loader = Zend_Context_Loader::factory("array","test.php");
Zend_Context::load($loader);
$log = Zend_Context::getDefaultObject("Zend_Log");
```

UC-05: Loading context from an array or a Zend_Config object

```
$data = array(
    "registry"=>"Zend_Registry"
    ,"components"=>array(
        "baseSql"=>array(
            "class"=>"Zend_Db"
            ,"default"=>"yes"
            ,"factory"=>"factory"
            ,"constructor"=>array(
                array("value"=>"PDO_Pgsql")
                ,array(
                    "value"=>array(
                         array("key"=>"host","value"=>"db.example.com")
                         ,array("key"=>"port","type"=>"int","value"=>"1234")
                         ,array("key"=>"username","value"=>"dbuser")
                         ,array("key"=>"password","value"=>"secret")
                         ,array("key"=>"dbname","value"=>"db")
                     )
                  )
              )
         )
     )
);
Zend_Context::load($data);
$db = Zend_Context::getObject("baseSql");

// clear the container
Zend_Context::reset();

$config = new Zend_Config($data);
Zend_Context::load($config);
$db = Zend_Context::getObject("baseSql");
```

UC-06: Using properties

```
../tests/Zend/Context/_files/test1.ini file:
webhost           = www.example.com
database.type     = pdo_mysql
database.host     = db.example.com
database.port     = 1234
database.username = dbuser
database.password = secret
database.name     = dbname
```

```
$data = array(
            "property"=>"contextProperty"
            ,"registry"=>"Zend_Registry"
            ,"components"=>array(
                "baseSql"=>array(
                    "class"=>"Zend_Db"
                    ,"default"=>"yes"
                    ,"factory"=>"factory"
                    ,"constructor"=>array(
                        array("value"=>"PDO_Pgsql")
                        ,array(
                            "value"=>array(
                                array("key"=>"host","value"=>"${database.host}")
                                ,array("key"=>"port","type"=>"init","value"=>"${database.port}")
                                ,array("key"=>"username","value"=>"${database.username}")
                                ,array("key"=>"password","value"=>"${database.password}")
                                ,array("key"=>"dbname","value"=>"${database.name}")
                            )
                        )
                    )
                )
                ,"contextProperty"=>array(
                    "class"=>"Zend_Context_Property"
                    ,"constructor"=>array(
                        array("value"=>"include_path:../tests/Zend/Context/_files/test1.ini")
                    )
                )
            )
            ,"aliases"=>array("pgSql"=>"baseSql")
        );
Zend_Context::load($data);
$db = Zend_Context::getObject("baseSql");
```

UC-07: Using components

```
$component = new Zend_Context_Component("baseSql","Zend_DB");
$kind = $component->createArgument("PDO_Pgsql","kind");
$config = $component->createArrayArgument(array(
    "host"     => 'db.example.com',
    "username" => 'dbuser',
    "password" => 'secret',
    "dbname"   => 'db'
),"config");
$db = $component->setDefault("yes")->setFactory("factory")
    ->addConstructorArgument($kind)->addConstructorArgument($config)->createInstance();
Zend_Context::addComponent($component);
```

9. Class Skeletons

```
class Zend_Context_Exception extends Zend_Exception {}
```

```
class Zend_Context {

    /** @var bool is context already loaded */
    protected static $_loaded = false;

    /** @var array the components loaded */
    protected static $_components = array();
```

```
    /** @var array the components in loading, to prevent a circle */
    protected static $_componentsLoading = array();

    /** @var array the properties components, that can import variables */
    protected static $_properties = array();

    /** @var mixed the registry implementation (array, Zend_Registry, ...) */
    protected static $_registry = array();

    /** @var array the default objects for classes */
    protected static $_defaults = array();

    /**
     * Load the context from a loader, Zend_Config or array
     *
     * @param  mixed the data (Zend_Context_Loader, Zend_Config, array)
     * @return void
     */
    public static function load($data);

    /**
     * Sets the registry's implementation
     *
     * @param  string the implemetation's classname ("Zend_Registry")
     * @return void
     */
    public static function setRegistry($registryClassName);

    /**
     * Returns the registry's implementation
     *
     * @return mixed the registry
     */
    public static function getRegistry();

    /**
     * Sets the properties for the context
     *
     * @param  mixed the property components' names separated with comma
     *               or an array of the components' names
     * @return void
     */
    public static function addProperties($properties);

    /**
     * Returns the properties components
     *
     * @return array array of the components
     */
    public static function getProperties();

    /**
     * Sets an alias for a component
     *
     * @param string the old component's name
     * @param string the alias name for the component
     * @return void
     */
    public static function setAlias($oldId,$newId);

    /**
     * Is the context already loaded
     *
     * @return bool is it loaded
     */
    public static function isLoaded();

    /**
     * Clears everything from context
```

```
     *
     * @return void
     */
    public static function reset();

    /**
     * Returns a component's object from the context
     *
     * @param  string the component's name
     * @return mixed the stored object
     */
    public static function getObject($id);

    /**
     * Returns the default component's object for a classname
     *
     * @param  string the classname
     * @return mixed the stored object
     */
    public static function getDefaultObject($className);

    /**
     * Returns a component from the context
     *
     * @param  string the component's name
     * @return Zend_Context_Component the component
     */
    public static function getComponent($id);

    /**
     * Returns a default component for a classname
     *
     * @param  string the classname
     * @return Zend_Context_Component the component
     */
    public static function getDefaultComponent($className);

    /**
     * Sets a default component for a classname
     *
     * @param string the component's name
     * @param string the classname
     * @return void
     */
    public static function setDefault($id,$className);

    /**
     * Adds a component to the context
     *
     * @param Zend_Context_Component the component
     * @return void
     */
    public static function addComponent(Zend_Context_Component $component);

    /**
     * Replace a component with a new one
     *
     * @param mixed the component's name or the component
     * @param Zend_Context_Component the new component
     * @return void
     */
    public static function replaceComponent($old,Zend_Context_Component $new);

    /**
     * Deletes a component from the context
     *
     * @param mixed the component's name or the component
     * @return void
     */
```

```
    public static function removeComponent($id);

}
```

```
abstract class Zend_Context_Loader {

    /**
     * Creates and returns a loader from the parameters
     *
     * @param string the loader's kind (array, xml)
     * @param mixed the data (it depends on the kind)
     * @param mixed the schema for checking the data (it depends on the kind)
     * @return mixed the loader (Zend_Context_Loader_Array or Zend_Context_Loader_Xml)
     */
    public static function factory($kind,$data,$schema=null) {'

    /**
     * Constructor
     *
     * @param mixed the data, that needs to be loaded
     * @return mixed the object
     */
    abstract protected function __construct($data);

    /**
     * Returns a loaded data
     *
     * @return mixed the data that can be read by the context
     */
    abstract public function get();

    /**
     * Validate the data
     *
     * @param mixed the data
     * @return bool is it valid
     */
    abstract public function validate($data);

}
```

```
class Zend_Context_Loader_Exception extends Zend_Context_Exception {}
```

```
class Zend_Context_Loader_Array extends Zend_Context_Loader {

    /** @var array the loaded data */
    protected $_data = array();

    /**
     * Constructor
     *
     * @param mixed the data, that needs to be loaded
     * @param mixed the schema to validate to
     * @return mixed the object
     */
    protected function __construct($data,$schema=null);

    /**
     * Returns a loaded data
     *
     * @return mixed the data that can be read by the context
     */
    public function get();

    /**
     * Validate the data
     *
     * @param mixed the data
     * @return bool is it valid
     */
    public function validate($data);
}
```

```
class Zend_Context_Loader_Xml extends Zend_Context_Loader {

    /** @var DomDocument the loaded xml */
    protected $_xml = null;

    /** @var array the loaded data */
    protected $_data = array();

    /**
     * Constructor
     *
     * @param mixed the data, that needs to be loaded
     * @param mixed the schema to validate to
     * @return mixed the object
     */
    protected function __construct($data,$schema=null);

    /**
     * Returns a loaded data
     *
     * @return mixed the data that can be read by the context
     */
    public function get();

    /**
     * Validate the data
     *
     * @param mixed the data
     * @param mixed the schema to validate to
     * @return bool is it valid
     */
    public function validate($data,$schema=null);

    /**
     * Converts the incoming data to DomDocument
     *
```

```
     * @param mixed the data (DomDocument, SimpleXMLElement, string)
     * @return void
     */
    protected function _convertDataToXml($data);

    /**
     * Parse a file or string
     *
     * @param string the incoming data or a filename that contains the data
     * @return string the parsed data
     */
    protected function _parseData($data);

    /**
     * Get any errors from validation
     *
     * @return string error's string
     */
    protected function _getValidationError();

    /**
     * Convert the xml to array
     *
     * @return array the array that can be read by context
     */
    protected function _convertXmlToArray();

    /**
     * Convert an xml node to component array
     *
     * @param DomNode the node
     * @return array the component array
     */
    protected function _convertComponent($node);

    /**
     * Convert an xml node to an alias array
     *
     * @param DomNode the node
     * @return array the alias array
     */
    protected function _convertAliases($node);

    /**
     * Convert an xml node's attributes to component attributes
     *
     * @param DomNode the node
     * @return array the attributes array
     */
    protected function _convertAttributes($node);

    /**
     * Convert an xml node's childnodes to component arguments
     *
     * @param DomNode the node
     * @return array the argument array
     */
    protected function _convertArguments($node);

    /**
     * Convert an xml node to an array
     *
     * @param DomNode the node
     * @return array the array
     */
    protected function _convertNodeToArray($node);

    /**
     * Gets the value of the node (nodeValue or a node named value)
     *
```

```
     * @param DomNode the node
     * @return string the value
```

```
     */
    protected function _getNodeValue($node);
}
```

```
interface Zend_Context_Component_Interface {

    /**
     * Gets the component's id
     *
     */
    public function getId();

    /**
     * Returns the component's classname
     *
     */
    public function getClass();

    /**
     * Returns if it's a default for a class
     *
     */
    public function getDefault();

    /**
     * Return the component converted to array
     *
     */
    public function toArray();

    /**
     * Sets the component's classname
     *
     * @param string
     */
    public function setClass($class);

    /**
     * Sets the component's factory methodname
     *
     * @param string
     */
    public function setFactory($factory);

    /**
     * Sets the component to be default for a class
     *
     * @param string ("yes","no")
     */
    public function setDefault($default);

    /**
     * Sets the component's init method (it gets called first after creating an instance)
     *
     * @param string
     */
    public function setInit($init);

    /**
     * Creates an argument for the component
     *
     * @param string the value
     * @param string the argument's name
     */
    public function createArgument($value,$name="");
```

```
/**
 * Creates an array argument for the component
 *
 * @param array the value
 * @param string the argument's name
 */
public function createArrayArgument($value,$name="");

/**
 * Creates a reference argument for the component
 *
 * @param string the reference's name
 */
public function createReferenceArgument($ref);

/**
 * Adds a constructor argument to the array
 *
 * @param array the argument
 */
public function addConstructorArgument($arg);

/**
 * Adds a method argument to the array
 *
 * @param array the argument
 */
public function addMethodArgument($name,$arg);

/**
 * Adds a setter argument to the array
 *
 * @param array the argument
 */
public function addSetterArgument($variableName,$arg);

/**
 * Creates an object from the component
 *
```

```
     */
    public function createInstance();
}
```

```
class Zend_Context_Component implements Zend_Context_Component_Interface {

    /** @var string the component's id */
    protected $_id;

    /** @var string the component's classname */
    protected $_class;

    /** @var string the component's factory method */
    protected $_factory = "";

    /** @var string is the component default for a class ("yes","no") */
    protected $_default = "no";

    /** @var string the component's init method, that gets called first after creating an instance */
    protected $_init = "";

    /** @var array the component's constructor arguments */
    protected $_constructor = array();

    /** @var array the component's methods */
    protected $_methods = array();

    /** @var array the component's setters */
    protected $_setters = array();

    /**
     * Constructor
     *
     * @param string the component's id
     * @param string the component's classname
     * @param string the component's factory methodname
     * @param string is the component default for a class ("yes","no")
     * @param string the component's init method, that gets called first after creating an instance
     * @return void
     */
    public function __construct($id,$class,$factory="",$default="no",$init="");

    /**
     * Gets the component's id
     *
     * @return string
     */
    public function getId();

    /**
     * Returns the component's classname
     *
     * @return string
     */
    public function getClass();

    /**
     * Returns if it's a default for a class
     *
     * @return string ("yes", "no")
     */
    public function getDefault();

    /**
     * Sets the component's classname
     *
```

```
     * @param string
     * @return void
     */
    public function setClass($class);

    /**
     * Sets the component's factory methodname
     *
     * @param string
     * @return void
     */
    public function setFactory($factory);

    /**
     * Sets the component to be default for a class
     *
     * @param string ("yes","no")
     * @return void
     */
    public function setDefault($default);

    /**
     * Sets the component's init method (it gets called first after creating an instance)
     *
     * @param string
     * @return void
     */
    public function setInit($init);

    /**
     * Creates an argument for the component
     *
     * @param string the value
     * @param string the argument's name
     * @return array the argument
     */
    public function createArgument($value,$name="");

    /**
     * Creates an array argument for the component
     *
     * @param array the value
     * @param string the argument's name
     * @return array the argument
     */
    public function createArrayArgument($array,$name="");

    /**
     * Creates a reference argument for the component
     *
     * @param string the reference's name
     * @return array the argument
     */
    public function createReferenceArgument($ref);

    /**
     * Adds a constructor argument to the array
     *
     * @param array the argument
     * @return Zend_Context_Component
     */
    public function addConstructorArgument($arg);

    /**
     * Adds a setter argument to the array
     *
     * @param array the argument
     * @return Zend_Context_Component
     */
    public function addSetterArgument($variableName,$arg);
```

```
/**
 * Adds a method argument to the array
 *
 * @param array the argument
 * @return Zend_Context_Component
 */
public function addMethodArgument($name,$arg);

/**
 * Creates an object from the component
 *
 * @return mixed
 */
public function createInstance();

/**
 * Return the component converted to array
 *
 * @return array
 */
public function toArray();

/**
 * Converts the arguments to real arguments to create an instance
 *
 * @param array the component's stored argument definition
 * @return array
 */
protected function _getArguments($argumentsDef);

/**
 * Converts the array arguments to real arguments to create an instance
 *
 * @param array the component's stored array argument definition
 * @return array
 */
protected function _getArrayArgument($argument);

/**
 * Convert the arguments value to a given type
 *
 * @param array the argument
 * @return mixed the converted value
 */
protected function _convertArgument($argument);

/**
 * Converts values with the context's properties to real values
 *
 * @param string a value
 * @return string
```

```
     */
    protected function _convertProperties($argument);
}
```

```
class Zend_Context_Property {

    /** @var array the property's locations */
    protected $_locations = array();

    /** @var string the property's kind (ini) */
    protected $_kind = null;

    /** @var array the property's data */
    protected $_data = null;

    /** @var bool has the property been loaded */
    protected $_loaded = false;

    /**
     * Constructor
     *
     * @param mixed string (comma separated) or an array of locations
     * @param string the kind of the property
     * @return void
     */
    public function __construct($locations=array(),$kind="ini");

    /**
     * Get the locations of the properties
     *
     * @param boolean if true, we get a comma separated string, if false we get an array
     * @return mixed (string, array)
     */
    public function getLocations($string=true);

    /**
     * Returns a value that was found in a property
     *
     * @param string
     * @return string
     */
    public function get($name);

    /**
     * Sets the locations of the properties
     *
     * @param mixed string (comma separated) or an array of locations
     * @return void
     */
    protected function _setLocations($locations);

    /**
     * Loads the properties from the locations
     *
     * @return void
     */
    protected function _load();

    /**
     * Gets the real locations of the properties
     *
     * @param string the location
     * @return string the real location
     */
    protected function _getRealLocation($location);

}
```

context.xsd:

```
<?xml version="1.0" encoding="UTF-8"?>

<xsd:schema xmlns="http://www.zend.com/schema/context"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://www.zend.com/schema/context">

  <xsd:element name="context">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="property" type="propertyRegistryType"
        maxOccurs="1" minOccurs="0"/>
        <xsd:element name="registry" type="propertyRegistryType"
        maxOccurs="1" minOccurs="0"/>
        <xsd:element name="components" type="componentsType"
        maxOccurs="1" minOccurs="0"/>
        <xsd:element name="aliases" type="aliasesType"
        maxOccurs="1" minOccurs="0"/>
      </xsd:sequence>
      <xsd:attribute name="property" type="xsd:string" use="optional"/>
      <xsd:attribute name="registry" type="xsd:string" use="optional"/>
    </xsd:complexType>
  </xsd:element>

  <xsd:complexType name="propertyRegistryType">
    <xsd:simpleContent>
      <xsd:extension base="xsd:string">
        <xsd:attribute name="value" type="xsd:string" use="optional"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>

  <xsd:complexType name="componentsType">
    <xsd:sequence>
      <xsd:element name="component" type="componentType"
      maxOccurs="unbounded" minOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="componentType">
    <xsd:sequence>
      <xsd:element name="id" type="xsd:string"
              maxOccurs="unbounded" minOccurs="0"/>
      <xsd:element name="class" type="xsd:string"
              maxOccurs="unbounded" minOccurs="0"/>
      <xsd:element name="default" type="defaultType"
              maxOccurs="unbounded" minOccurs="0"/>
      <xsd:element name="factory" type="xsd:string"
              maxOccurs="unbounded" minOccurs="0"/>
      <xsd:element name="init" type="xsd:string"
              maxOccurs="unbounded" minOccurs="0"/>
      <xsd:element name="constructor" type="functionType"
              maxOccurs="unbounded" minOccurs="0"/>
      <xsd:element name="setters" type="settersType"
              maxOccurs="unbounded" minOccurs="0"/>
      <xsd:element name="methods" type="methodsType"
              maxOccurs="unbounded" minOccurs="0"/>
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:string" use="optional"/>
    <xsd:attribute name="class" type="xsd:string" use="optional"/>
    <xsd:attribute name="default" type="defaultType" use="optional"/>
    <xsd:attribute name="factory" type="xsd:string" use="optional"/>
    <xsd:attribute name="init" type="xsd:string" use="optional"/>
  </xsd:complexType>

  <xsd:simpleType name="defaultType">
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="yes|no"/>
    </xsd:restriction>
  </xsd:simpleType>
```

```
<xsd:complexType name="settersType">
  <xsd:sequence>
    <xsd:element name="setter" type="setterType"
    maxOccurs="unbounded" minOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="methodsType">
  <xsd:sequence>
    <xsd:element name="method" type="functionType"
    maxOccurs="unbounded" minOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="aliasesType">
  <xsd:sequence>
    <xsd:element name="alias" type="elementType"
    maxOccurs="unbounded" minOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="argumentType" mixed="true">
  <xsd:sequence>
    <xsd:element name="array" type="arrayType"
    maxOccurs="unbounded" minOccurs="1"/>
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string" use="optional"/>
  <xsd:attribute name="value" type="xsd:string" use="optional"/>
  <xsd:attribute name="ref" type="xsd:string" use="optional"/>
  <xsd:attribute name="type" type="typeType" use="optional"/>
</xsd:complexType>

<xsd:complexType name="functionType">
  <xsd:sequence>
    <xsd:element name="arg" type="argumentType"
    maxOccurs="unbounded" minOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="setterType" mixed="true">
  <xsd:sequence>
    <xsd:element name="array" type="arrayType"
    maxOccurs="unbounded" minOccurs="1"/>
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string" use="required"/>
  <xsd:attribute name="value" type="xsd:string" use="optional"/>
  <xsd:attribute name="ref" type="xsd:string" use="optional"/>
  <xsd:attribute name="type" type="typeType" use="optional"/>
</xsd:complexType>

<xsd:complexType name="arrayType">
  <xsd:sequence>
    <xsd:element name="element" type="elementType"
    maxOccurs="unbounded" minOccurs="1"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="elementType">
  <xsd:simpleContent>
    <xsd:extension base="xsd:string">
      <xsd:attribute name="value" type="xsd:string" use="optional"/>
      <xsd:attribute name="key" type="xsd:string" use="required"/>
      <xsd:attribute name="type" type="typeType" use="optional"/>
    </xsd:extension>
  </xsd:simpleContent>
</xsd:complexType>

<xsd:simpleType name="typeType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="int" />
```

```
    <sd:enumeration value="float" />
    <sd:enumeration value="bool" />
    <sd:enumeration value="object" />
    <sd:enumeration value="array" />
    <sd:enumeration value="string" />
  </sd:restriction>
```

```
    </xsd:simpleType>

  </xsd:schema>
```

]]></ac:plain-text-body></ac:macro>
]]></ac:plain-text-body></ac:macro>